Computer Graphics Lab 1 & 2 Report

一、实验内容

1.多边形填充扫描线算法与加权反走样的实现。

2.二维线段裁剪 Cohen-Sutherland 算法的实现。

3.二维线段裁剪 Liang-Barsky 算法的实现。

二、测试环境

OS: openSUSE 42.2 64Bit

CPU: Intel Core i7-6500U @ 2.50GHz

RAM: 8GB DDR4 2133MHz

Compiler Version: g++ (GCC) 6.3.0

三、实现过程

1. 多边形填充扫描线算法与加权反走样:

以下是图形属性设置:

```
10
11 const int maxX = 1080, maxY = 1920, maxP = 10;
12
```

maxX 和 maxY 决定图形的长和宽，maxP 是随机生成的点的个数。该程序随机生成 maxP 个点，依次连接构成多边形（不一定是简单多边形），然后使用扫描线算法填充。

```
124
125                 for (int j = 0; j < allveEdge.size(); j++)
126                         if (allveEdge[j].yMax <= yCurr)
127                                 allveEdge[j].alive = false;
128
129                 for (int j = 0; j < allveEdge.size(); j++)
130                         allveEdge[j].xCurr += 0.2 * allveEdge[j].delta;
131
132                 while (edgeCurr < maxP && edge[edgeCurr].yMin <= yCurr) {
133                         Edge tempEdge = edge[edgeCurr];
134                         tempEdge.xCurr += (yCurr - edge[edgeCurr++].yMin) * tempEdge.delta;
135                         allveEdge.push_back(tempEdge);
136                 }
137
```

使用 vector 实现活性边表，同时使用 LazyTag 实现 O(1)删除。

以下是使用高斯分布函数计算得来的反走样加权矩阵:

```
46 const int val[5][5] = {
47         1, 2, 4, 2, 1,
48         2, 5, 6, 5, 2,
49         4, 6, 8, 6, 4,
50         2, 5, 6, 5, 2,
51         1, 2, 4, 2, 1,
52 }, SUM = 88;
```

该程序将一个像素细分成 5*5 的矩阵，并对其进行填充，而后根据其填充情况决定该像素的灰度值。

2. 二维裁剪的 Cohen-Sutherland 算法

```
12 const int LineW = 2;
13 const int xMax = 2500, xMin = 2000, yMax = 2700, yMin = 2200;
14 //const int xMax = maxX, xMin = 0, yMax = maxY, yMin = 0;
15
```

使用 LineW 规定线段宽度，(xMin, xMax)和(yMin, yMax)规定了裁剪框的属性。

```
49
50 void Cohen_SutherLand(Point& a, Point& b) {
51         int aOrder = 0, bOrder = 0;
52
53         if (a.x < xMin)
54                 aOrder |= (1 << 0);
55         if (a.x > xMax)
56                 aOrder |= (1 << 1);
57         if (a.y < yMin)
58                 aOrder |= (1 << 2);
59         if (a.y > yMax)
60                 aOrder |= (1 << 3);
61
62         if (b.x < xMin)
63                 bOrder |= (1 << 0);
64         if (b.x > xMax)
65                 bOrder |= (1 << 1);
66         if (b.y < yMin)
67                 bOrder |= (1 << 2);
68         if (b.y > yMax)
69                 bOrder |= (1 << 3);
70
```

根据线段端点 a, b 的位置信息将其编号。

```
70
71         if ((aOrder & bOrder) != 0) {
72                 a = Point(-1, -1);
73                 b = Point(-1, -1);
74                 return;
75         }
76
77         if ((aOrder | bOrder) == 0)
78                 return;
79
```

首先判断线段是否完全在裁剪框内外。

```
79
80          if (aOrder != 0) {
81                  if ((aOrder & (1 << 0)) != 0) {
82                          a.y = a.y + (b.y - a.y) / (b.x - a.x) * (xMin - a.x);
83                          a.x = xMin;
84                          Cohen_SutherLand(a, b);
85                  }
86                  else if ((aOrder & (1 << 1)) != 0) {
87                          a.y = a.y + (b.y - a.y) / (b.x - a.x) * (xMax - a.x);
88                          a.x = xMax;
89                          Cohen_SutherLand(a, b);
90                  }
91                  else if ((aOrder & (1 << 2)) != 0) {
92                          a.x = a.x + (b.x - a.x) / (b.y - a.y) * (yMin - a.y);
93                          a.y = yMin;
94                          Cohen_SutherLand(a, b);
95                  }
96                  else if ((aOrder & (1 << 3)) != 0) {
97                          a.x = a.x + (b.x - a.x) / (b.y - a.y) * (yMax - a.y);
98                          a.y = yMax;
99                          Cohen_SutherLand(a, b);
100                 }
101         }
102         else if (bOrder != 0) {
103                 if ((bOrder & (1 << 0)) != 0) {
104                         b.y = b.y + (a.y - b.y) / (a.x - b.x) * (xMin - b.x);
105                         b.x = xMin;
106                         Cohen_SutherLand(a, b);
107                 }
108                 else if ((bOrder & (1 << 1)) != 0) {
109                         b.y = b.y + (a.y - b.y) / (a.x - b.x) * (xMax - b.x);
110                         b.x = xMax;
111                         Cohen_SutherLand(a, b);
112                 }
113                 else if ((bOrder & (1 << 2)) != 0) {
114                         b.x = b.x + (a.x - b.x) / (a.y - b.y) * (yMin - b.y);
115                         b.y = yMin;
116                         Cohen_SutherLand(a, b);
117                 }
118                 else if ((bOrder & (1 << 3)) != 0) {
119                         b.x = b.x + (a.x - b.x) / (a.y - b.y) * (yMax - b.y);
120                         b.y = yMax;
121                         Cohen_SutherLand(a, b);
122                 }
123         }
124
```

判断线段哪一端点在裁剪框之外，将其裁剪后递归调用 CS 算法。

3. 二维裁剪的 Liang-Barsky 算法

```
127
128 void LiangYoudong_Barsky(Point& a, Point& b) {
129         if (a.x == b.x) {
130                 if (a.y >= b.y)
131                         swap(a, b);
132
133                 if (a.x > xMax || b.x < xMin) {
134                         a = Point(-1, -1);
135                         b = Point(-1, -1);
136                 }
137                 else if (a.y > yMax || a.y < yMin) {
138                         a = Point(-1, -1);
139                         b = Point(-1, -1);
140                 }
141                 else {
142                         a.y = max(a.y, (double) yMin);
143                         b.y = min(b.y, (double) yMax);
144                 }
145
146                 return;
147         }
148
149         if (a.y == b.y) {
150                 if (a.x >= b.x)
151                         swap(a, b);
152
153                 if (a.x > xMax || b.x < xMin) {
154                         a = Point(-1, -1);
155                         b = Point(-1, -1);
156                 }
157                 else if (a.y > yMax || a.y < yMin) {
158                         a = Point(-1, -1);
159                         b = Point(-1, -1);
160                 }
161                 else {
162                         a.x = max(a.x, (double) xMin);
163                         b.x = min(b.x, (double) xMax);
164                 }
165
166                 return;
167         }
```

首先处理水平和竖直的直线。

```
168
169         double t_xMin = (xMin - a.x) / (b.x - a.x),
170                t_xMax = (xMax - a.x) / (b.x - a.x),
171                t_yMin = (yMin - a.y) / (b.y - a.y),
172                t_yMax = (yMax - a.y) / (b.y - a.y);
173
174         if (t_yMin > t_yMax)
175                 swap(t_yMin, t_yMax);
176         if (t_xMin > t_xMax)
177                 swap(t_xMin, t_xMax);
178
179         double t_Begin = max(0.0, max(t_xMin, t_yMin)),
180                t_End = min(1.0, min(t_xMax, t_yMax));
181
182         if (t_Begin >= t_End) {
183                 a = Point(-1, -1);
184                 b = Point(-1, -1);
185         }
186         else {
187                 Point aTemp = a, bTemp = b;
188
189                 a.x = aTemp.x + t_Begin * (bTemp.x - aTemp.x);
190                 a.y = aTemp.y + t_Begin * (bTemp.y - aTemp.y);
191
192                 b.x = aTemp.x + t_End * (bTemp.x - aTemp.x);
193                 b.y = aTemp.y + t_End * (bTemp.y - aTemp.y);
194         }
195
```

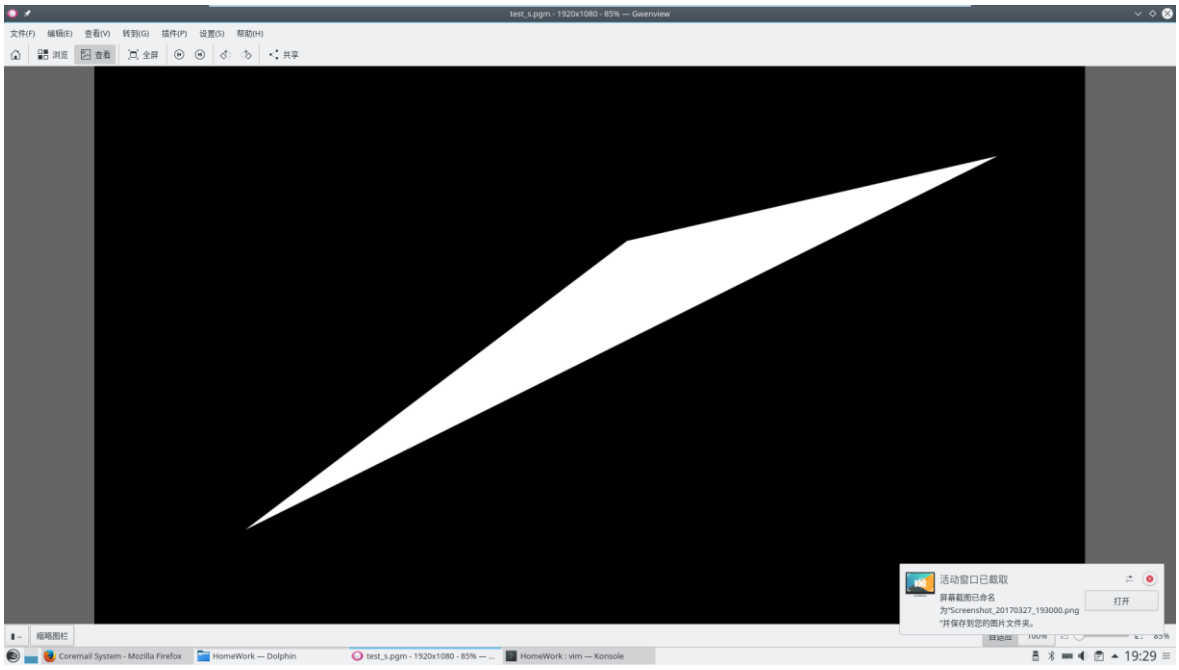
而后将其转化为参数表示，并求出在裁剪框内的参数范围。

四、结果示例

注：输出以 PGM 灰度图片格式编码，所提交程序均会自动随机取点绘图。

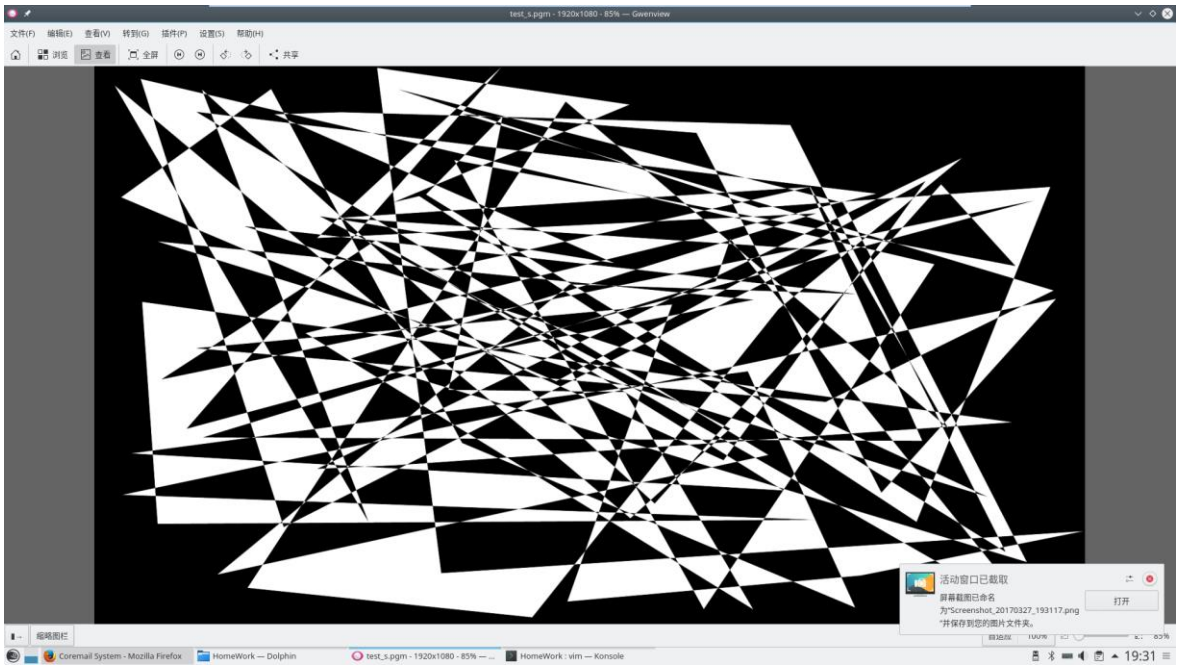
1.多边形填充与反走样

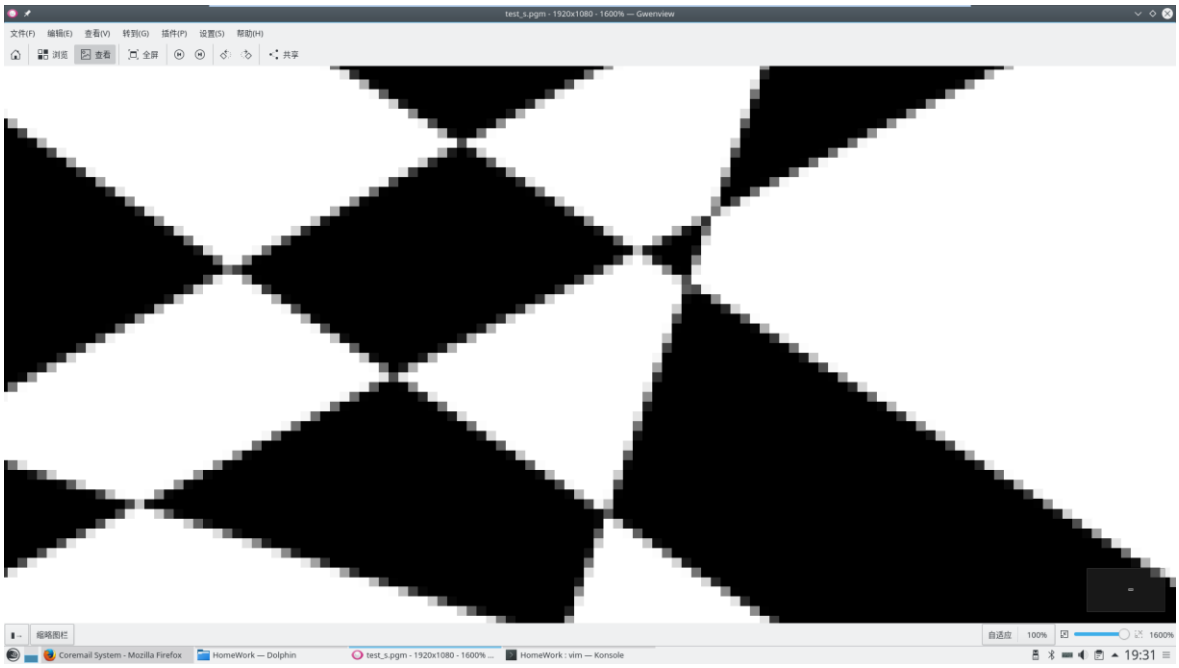
随机三角形：



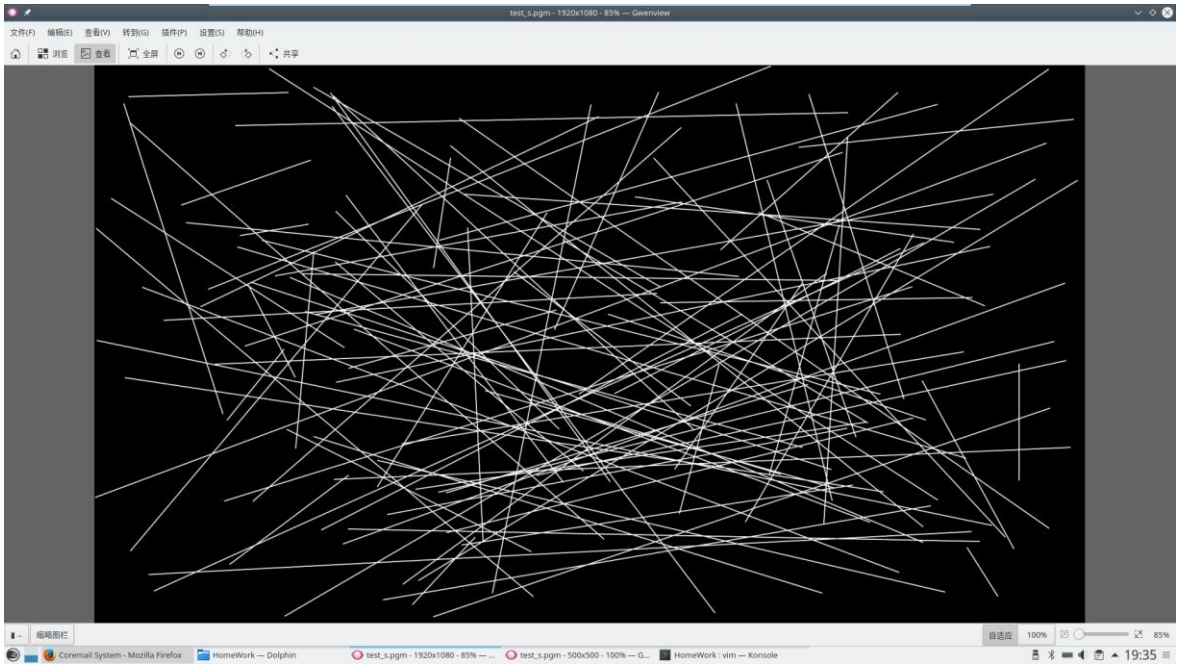

test_s.pgm - 1920x1080 - 1600% — Gwenview
文件(F) 编辑(E) 查看(V) 转到(G) 插件(P) 设置(S) 帮助(H)
浏览 查看 全屏 共享
缩略图栏
自适应 100% 1600%
Coremail System - Mozilla Firefox
HomeWork — Dolphin
test_s.pgm - 1920x1080 - 1600% ...
HomeWork : vim — Konsole
19:30

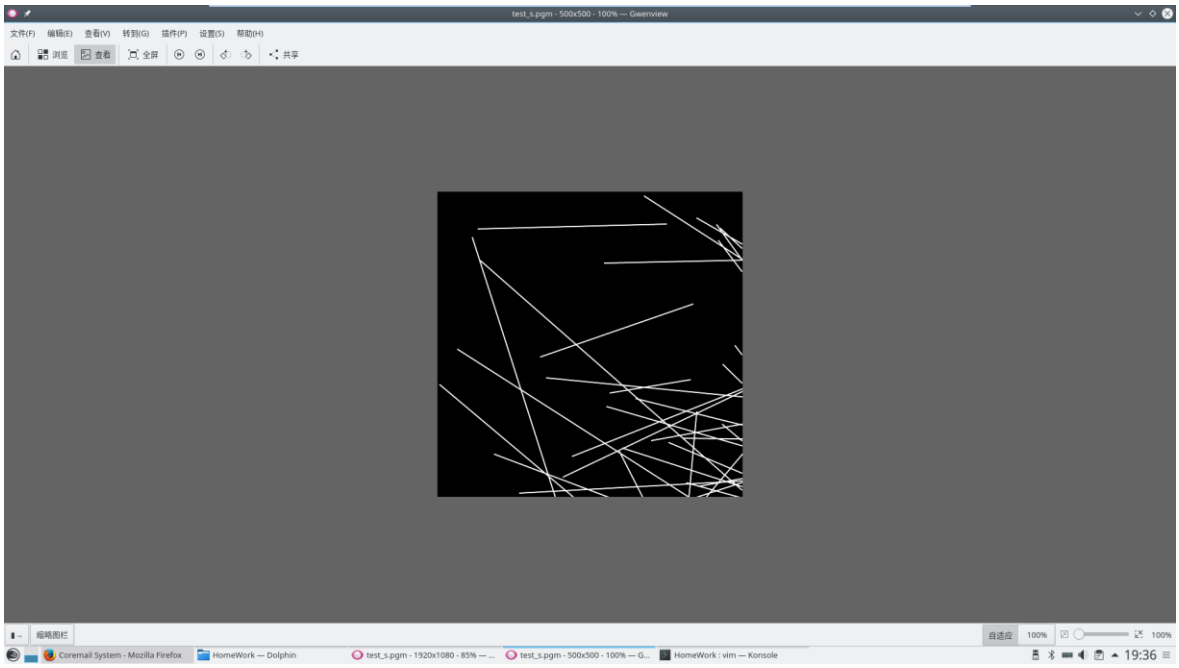
随机 100 边形：


test_s.pgm - 1920x1080 - 85% — Gwenview
文件(F) 编辑(E) 查看(V) 转到(G) 插件(P) 设置(S) 帮助(H)
浏览 查看 全屏 共享
活动窗口已截取
屏幕截图已命名
为"Screenshot_20170327_193117.png
"并保存到您的图片文件夹。
打开
缩略图栏
Coremail System - Mozilla Firefox
HomeWork — Dolphin
test_s.pgm - 1920x1080 - 85% — ...
HomeWork : vim — Konsole
19:31

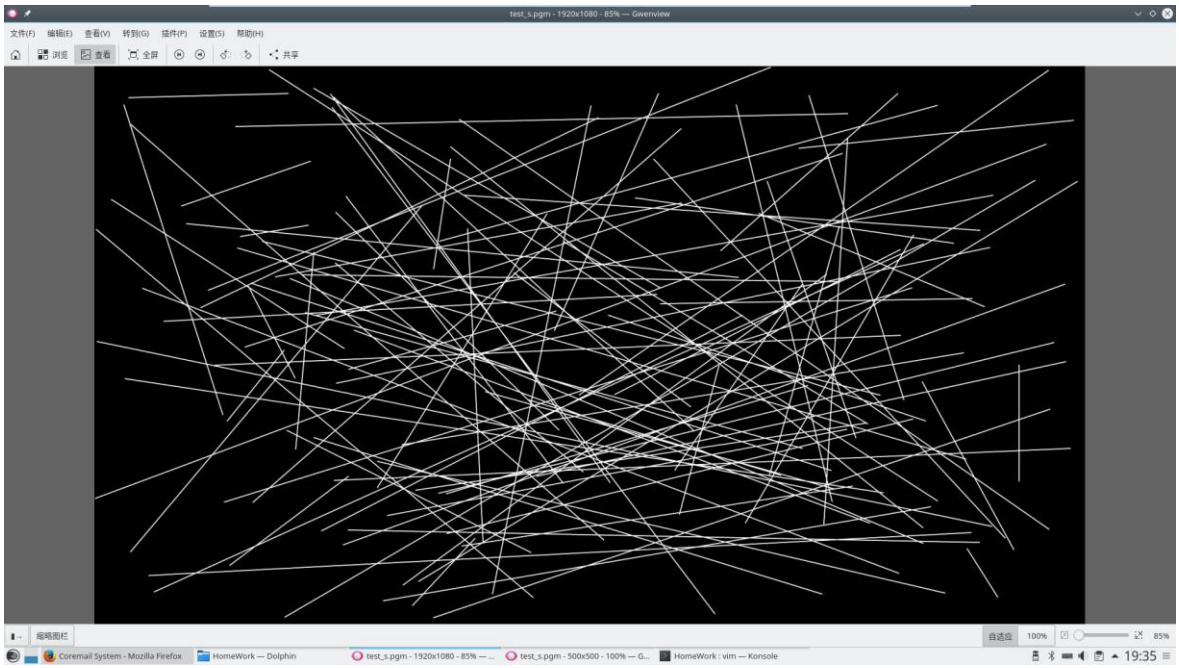


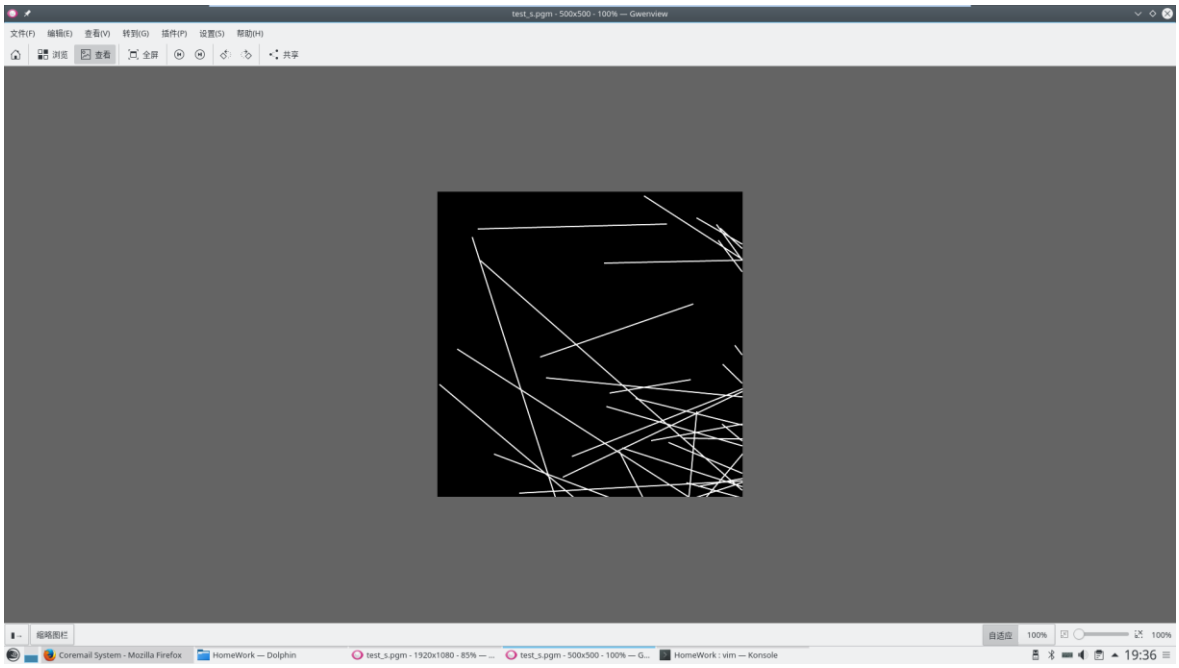
2.二维线段裁剪的 Cohen-Sutherland 算法





3.二维线段裁剪的 Liang-Barsky 算法




test_s.pgm - 500x500 - 100% — Gwenview
文件(F)
编辑(E)
查看(V)
转到(G)
插件(P)
设置(S)
帮助(H)
浏览
查看
全屏
共享
缩略图栏
自适应
100%
100%
Coremail System - Mozilla Firefox
HomeWork — Dolphin
test_s.pgm - 1920x1080 - 85% — ...
test_s.pgm - 500x500 - 100% — G...
HomeWork : vim — Konsole
19:36